[MAC0211] Laboratório de Programação I
Aula 12
Arquivos – Filtros – Interfaces

Kelly Rosa Braghetto

DCC–IME–USP

16 de abril de 2013

[Aula passada] Arquivos de dispositivos

O Unix e seus derivados são sistemas orientados a arquivos:

- Os dispositivos periféricos também são tratados como um tipo especial de arquivo
- Esses arquivos especiais possibilitam que programas interajam com qualquer dispositivo por meio de chamadas ao sistema padronizadas para operações de E/S
- Exemplos de arquivos de dispositivos de entrada e saída:
  - impressora – /dev/lp0
  - console – /dev/console
  - hard disk – /dev/sd[x] ou /dev/hd[x]...
  - cdrom – /dev/cdrom
  - /dev/null [não está associado a um dispositivo físico!]
- Arquivos de dispositivos podem ser de três tipos: *de caracter*, *de bloco* e *pseudo-dispositivo*

"Brincando" com os arquivos de dispositivos

Faça este teste e observe o resultado:

- em um terminal, executar o comando `tty`. O resultado será algo como `/dev/pts/3`.
- em outro terminal, executar o comando
  `cat arq >/dev/pts/3`
  (onde `arq` é um arquivo texto presente no diretório atual).
- Executar também
  `cat >/dev/pts/3`
  Digite caracteres e pressione [CRTL-D] para efetuar a "transmissão". [CRTL+D] em uma linha vazia encerra a "transmissão".

"Curiosidade" sobre o cat

- Quando nenhum arquivo de entrada é passado para o cat, ele lê caracteres da entrada padrão até que as teclas **[CRTL-D]** sejam pressionadas em uma linha vazia.
- Exemplo: o comando abaixo cria um arquivo de nome "meu_arq.txt", gravando nele tudo o que o usuário digitar até que **[CTRL-D]** seja pressionado em uma linha vazia

```
cat > meu_arq.txt
```

Streams

- No Unix e seus derivados, um *stream* é um fluxo de dados (bytes ou caracteres) , que pode ser tanto a entrada quanto a saída de um programa
- No fluxo de dados, os dados são acessados sequencialmente, um a um
- Todo processo tem ao menos três *streams*, os chamados

	0	Entrada padrão
streams de E/S padrão:	1	Saída padrão
	2	Saída de erro

- Geralmente, os *streams* da entrada padrão e saída padrão estão conectados ao teclado e ao monitor, respectivamente
- Os *shells* dos derivados do Unix possuem também as operações de redirecionamento, que conectam os *stream* de entrada e saída padrão a arquivos

Streams em C

- Antes de poder ler ou escrever em um arquivo, é preciso estabeler com ele uma conexão (ou canal de comunicação). Isso é feito na abertura do arquivo.
- Existem dois mecanismos diferentes para se representar conexões com arquivos: os *descritores de arquivos* (objetos do tipo `int`) e os *streams* (objetos do tipo `FILE*`)
- O conjunto de funções que realizam operações de escrita e leitura é muito mais rico e poderoso para *streams*
- Para descritores de arquivos, as funções se limitam a transferência de blocos de bytes
- Para *streams*, existem funções para a leitura e escrita formatada (`printf` e `scanf`), além de funções específicas para a leitura e escrita de caracteres e *strings* (`fgetc`, `fputs`, `getline`, etc.)

Pipes – mais detalhes sobre o seu funcionamento

- *Pipes* dependem da convenção de que todo programa tem inicialmente disponível para si (pelo menos) dois *streams* de E/S: a entrada padrão e a saída padrão
- O operação de *pipe* conecta a saída padrão de um programa à entrada padrão de outro. A cadeia de programas conectados desta forma é chamada de *pipeline*
- *Pipes* são um mecanismo de comunicação inter-programas

Filtros

- Ao implementar programas de modo que eles possam "conversar" entre si, evitamos a necessidade de implementar sistemas monolíticos (demasiadamente intrincados)
- A tradição Unix encoraja o desenvolvimento de programas que leiam e escrevam dados textuais sequencialmente, em formatos independentes de dispositivos
- Muitos programas do Unix e seus derivados são implementados como **filtros** simples, que recebem como entrada um *stream* de texto e processa-o, gerando um outro *stream* de texto como saída
- Exemplos de filtros: `cat`, `grep`, `wc`, ...

A filosofia do Unix

> *This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.*

de Doug McIlroy, inventor do conceito de *pipes* do Unix (trecho presente no livro *The Art of Unix Programming*, de E. S. Raymond)

Exemplo de filtro implementado em C

ROT13 – programa de encriptação [1]

```
#include <stdio.h>

/* Filtro que encripta um stream de texto  com o ROT13  */
int main(){
    int c;
    while ((c=getchar()) != EOF){
        if (c >= 'a' && c <= 'z')
            c = 'a' + (c - 'a' + 13) % 26;

        if (c >= 'A' && c <= 'Z')
            c = 'A' + (c - 'A' + 13) % 26;

        putchar(c);
    }
    return 0;
}
```

[1] Veja explicação em: http://pt.wikipedia.org/wiki/ROT13

Projeto de software

Essência do "projetar um software"

- Equilibrar objetivos e restrições concorrentes

Questões a serem trabalhadas no projeto

- Interfaces – quais são os serviços e acessos fornecidos?
- Ocultamento de informações – quais informações são visíveis e quais são privadas?
- Gerenciamento de recursos – quem é responsável por gerenciar memória e outros recursos limitados?
- Tratamento de erros – quem detecta os erros, quem os reporta, e como?

Projeto de interfaces

> *"Interfaces entre usuários, programas e partes de programas são fundamentais na programação e grande parte do sucesso de um software é determinado por quão bem as interfaces são projetadas e implementadas."*

Pike e Kernighan, no livro "A Prática da Programação"

Princípios de uma interface

Para ser bem sucedida, uma interface precisa ser bem adaptada à sua tarefa. Ela deve ser:

- Unificada – possuir um tema que unifique suas funções
- Simples – procurar esconder a complexidade de suas implementações
- Suficiente – prover as funcionalidades necessárias para satisfazer as necessidades dos usuários
- Genérica – ser suficientemente flexível para atender as necessidades de diferentes tipos de usuáris
- Estável – manter a estrutura e efeito de suas funções, mesmo quando as implementações são modificadas

Princípios de uma interface

Para projetar boas interfaces, siga o seguinte conjunto de princípios:

- oculte os detalhes de implementação (= encapsulamento, abstração, modularização)
- escolha um conjunto ortogonal pequeno de funções
- não saia do alcance do usuário
- faça uma mesma coisa igual em todos os lugares

Princípios de uma interface

Ocultação dos detalhes de implementação

A implementação por trás de uma interface deve ficar oculta, de modo que ela possa mudar sem afetar (negativamente) os sistemas que a usam.

Dicas:

- Evite o uso de variáveis globais; sempre que possível, é melhor passar os dados por meio de parâmetros para funções
- Não use dados que estão sempre "visíveis"; é difícil manter a consistência dos valores quando usuários podem alterar variáveis de forma indiscriminada
- Classes (de orientação a objetos) são um ótimo mecanismo para esconder informações

Princípios de uma interface

Escolha de um conjunto ortogonal pequeno de funções

A interface deve prover tantas funcionalidades quanto o necessário.
Funções não devem se sobrepor excessivamente no que se refere a suas funcionalidades.

Observações:

- Ter muitas funções pode tornar uma biblioteca mais fácil de ser usada, mas mas difícil de ser escrita e mantida
- Interfaces enormes são difíceis de ser "aprendidas" pelos usuários
- Não caia na tentação de incluir em sua interface funções que forneçam formas variadas de se fazer a mesma coisa (ex.: funções da libc para a escrita de um caracter em um stream – `putc`, `fputc`, `fprintf`, `fwrite`)
- Lembre-se da filosofia do Unix: "faça uma só coisa e faça-a bem feita". Não adicione itens a uma interface porque é possível fazê-lo e nem adicione-os para corrigir falhas na implementação

Princípios de uma interface

Não sair do alcance do usuário

- Uma biblioteca não deve criar variáveis de ambiente ou arquivos secretos, nem mudar dados globais
- Ela deve ser cuidadosa ao mudar dados de seu chamador
- Uma biblioteca não deve requerer outra apenas para a conveniência do projetista da interface ou do implementador
- É desejável que a biblioteca seja auto-contida. Quando isso não for possível, é preciso deixar explícito os serviços externos que ela requer

Princípios de uma interface

Fazer uma mesma coisa igual em todos os lugares

Consistência e regularidade são importantes.

- Coisas relacionados devem ser alcançadas por meio relacionados.
  Bom exemplo: funções básicas `str...` de C se comportam de forma parecida – dados fluem da esquerda para direita nos parâmetros e todas devolvem a *string* resultante).
  Mau exemplo: funções da biblioteca padrão de E/S de C, em que é difícil prever a ordem dos parâmetros para as funções – algumas possuem o parâmetro `FILE*` primeiro; outras, por último.
- Consistência externa de comportamento também deve ser um objetivo.
  Bom exemplo: o parâmetro de linha de comando `-v` geralmente habilita o modo "verboso" do programa chamado.
  Mau exemplo: *Browsers web* abrem um *link* com apenas em clique de mouse; outras aplicações demandam dois cliques para que um programa ou um *link* sejam carregados.

Gerenciamento de recursos

- É um dos problemas mais difíceis de se lidar no projeto de interfaces
- Se refere a: como gerenciar recursos que são de propriedade da biblioteca ou que são compartilhados pela biblioteca com aqueles que vão chamá-la?
- Exemplos de recursos: **memória**, arquivos, estado de variáveis
- Problemas: inicialização, manutenção do estado, compartilhamento e cópia, e limpeza

Gerenciamento de recursos

A liberação de um recurso deve ser feita na mesma camada em que ele foi alocado

- O estado da alocação de um recurso não deve ser alterado em toda a interface
- Ex1.: se uma função da interface recebe como entrada um arquivo aberto, então ela deve deixá-lo aberto quando for encerrada
- Os construtores e destrutores (de orientação a objetos) auxiliam na implantação dessa regra
- Ex2.: gerenciamento de memória com *garbage collector* (coleta automática de "lixo")

Tratamento de erros

O que fazer na ocorrência de um erro irrecuperável?

- Em muitos casos, mostar uma mensagem contendo detalhes sobre o erro ocorrido e sair do programa já é um tratamento apropriado
- Em outros casos, a melhor abordagem é apenas assinalar o erro e dar uma chance ao chamador de se recuperar
- Em alguns casos, nem mesmo mostrar uma mensagem é possível, porque bibioteca pode estar sendo executada em um ambiente em que a mensagem interferiria nos dados mostrados pelo chamador. Nesse caso, a melhor alternativa é registrar a mensagem de erro em um arquivo de *log*

Tratamento de erros

Detectar erros num nível baixo; lidar com eles num nível alto

- Ideia geral: o chamador é quem deve determinar a forma como o erro deve ser tratado
- As rotinas da biblioteca precisam colaborar com essa ideia: em casos de erro, devem falhar de forma "graciosa", ou seja, não abortando o código e retornando detalhes suficientes sobre o erro, para que o chamador possa fazer um tratamento apropriado
- Ex.: a função `getchar` devolve algum valor que não é caracter (como o EOF) quando o fim do arquivo é encontrado ou em caso de erro

Tratamento de erros

Usar exceções somente nas situações excepcionais

Algumas linguagens (como Java) possuem o conceito de exceções para capturar situações não usuais e se recuperar delas. Elas permitem que um fluxo de controle alternativo seja executado quando algo errado aconteça.

Dicas:

- Exceções não devem ser usadas para tratar valores de retorno esperados (como o EOF na leitura de um arquivo)
- Como elas distorcem o fluxo de controle, exceções podem conduzir a construções confusas e mais susceptíveis a erros

Interfaces com usuários

- Erros devem ser detectados e reportados; a recuperação deve ser tentada quando cabível
- Mensagens de erro devem ser tão informativas quanto possível (indicando a causa do erro)
- O texto das mensagens de erro, do *prompt* e das caixas de diálogo devem expressar o formato dos dados de entrada válidos (de forma a conduzir o usuário ao modo correto de uso do programa)
- Princípios de estilo que contribuem para a criação de interfaces (textuais ou gráficas) fáceis de se usar: simplicidade, clareza, regularidade, familiaridade, restrição.

Bibliografia e materiais recomendados

- Arquivos e Streams em C – manual de referência da libc
  `http://www.gnu.org/software/libc/manual/html_mono/libc.html#I_002f0-Overview`
- Filosofia do Unix e *pipes* – livro: *The Art of Unix Programming*, de E.S. Raymond
  `http://www.catb.org/esr/writings/taoup/html/`
- Capítulo 4 – *Interfaces*, do livro *A Prática da Programação*, de B.W. Kernighan e R. Pike
  (disponível no Paca, na página do curso)
- Capítulos 7 (*Libraries and Interfaces*) e 8 (*Designing Interfaces*) do livro *The Art and Science of C*, de E.S. Roberts
- Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
  `http://www.ime.usp.br/~kon/MAC211`

Cenas dos próximos capítulos...

Na próxima aula:

- Exemplo de interface
- Gerenciamento de compilação de programas e bibliotecas com ferramentas